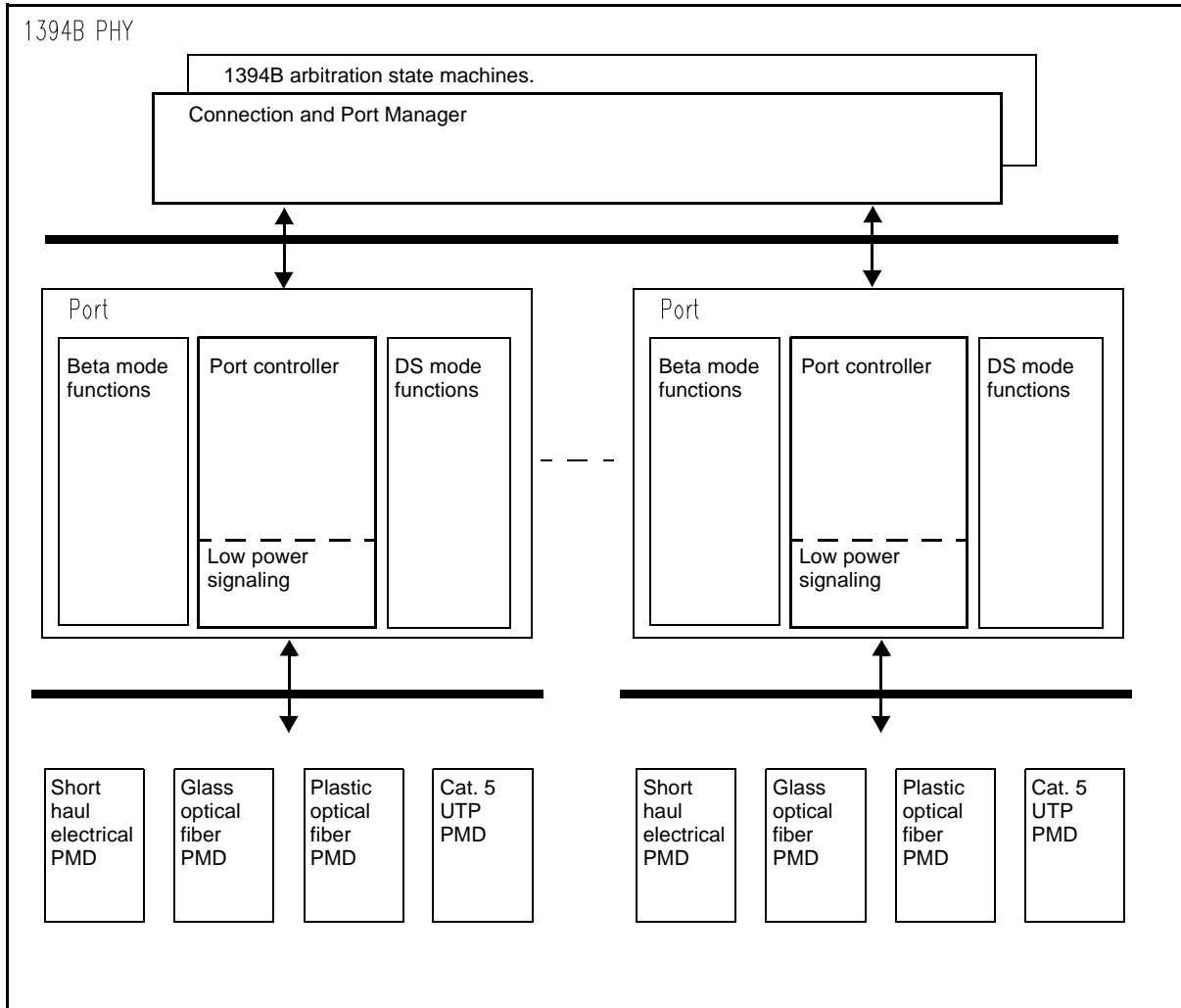


11. Connectivity Management

11.1 Overview

This section specifies the Connection and Port Manager for the physical layer and the Port Controller for each port, together with the communications used between these two modules, and the services provided by these two modules. The port abstraction is the primary interface inside the node between IEEE Std. 1394-1995 protocols and the P1394b gigabit protocols

The model for a port is shown below



The Connection Management function interacts with the physical media dependent layers, the Beta mode functions, DS mode functions and the repeater function by appropriate shared variables.

11.2 Port characteristics

The port is the device that controls the flow of data bits onto and off of the 1394 wires. The port state machine must detect the present or absence of its neighbor port, establish reliable communications with the other port in the most suitable mode of signaling, and must manage low-power modes of operation, i.e., sleep modes.

11.2.1 Definitions

- Port and port state machine are interchangeable terms when the context make the choice clear
- DS mode means the normal IEEE Std. 1394-1995 form of electrical signaling and handshaking
- Beta mode means the 8b/10b, full duplex form of binary encoded signaling
- Connection means the cabled interconnect between two ports
- Plug Present for a port means that there is a plug present at the local end. It does not imply an end-to-end connection. However, when no plug is present, further power savings are possible.
- Low Power signalling means signalling, based on the exchange of very low duty cycle tones, by which the connectivity status of a port is determined. This is used then the port is not active or disabled.
- Suspend means go into a low power mode of operation while maintaining the ability to process low power connection signalling
- Resume is a signal requiring the port to resume full speed, full power operation

11.2.2 Requirements

- a) The port must start up in DS mode if either the port or the port to which it is connected is not beta capable
- b) The port must start up in beta mode if both the ports are beta capable
- c) The port must run (the "operational speed") at the fastest speed of which both ports are capable, or allowed
- d) The port must support all speeds of packet at or below its operational speed through the use of data padding
- e) The port must support Suspend, Resume, and low power connection signalling.
- f) The port must detect and manage all connect/disconnect events

11.2.3 Port properties

A P1394b port

- a) shall be capable of operating at any speed range from S100 upwards, or any speed range from S800 upwards
- b) shall be capable of operating in Beta mode
- c) when capable of operating at S100, shall be capable of either DS at S100-S400, or Beta mode at S100-S400, or both
- d) shall not be brought out to a 1394-1995 connector unless capable of operating in DS mode
- e) may be connected directly to a suitable transceiver for long haul connection (thus also providing DC isolation)
- f) may use DC (e.g. capacitive or galvanic) isolation when operating in Beta mode

11.3 Connection Management

11.3.1 Connection Management Services

Connection management services are provided at the interface between the CM function and the Port Controller and/or the repeater function. The method by which these services are communicated is not defined by this standard.

11.3.1.1 CM status indication (CM_STATUS.indication)

The CM function uses this service to indicate the status of the port to the repeater function. The following parameters are communicated via this service:

- 1 a) Port mode. This parameter shall contain the mode of the port. The following values are defined for this parameter:
- 2 1) DS. The port is operating in DS mode.
- 3 2) BETA. The port is operating in beta mode.
- 4 b) Port speed. This parameter shall contain the operating speed of the port. The value of this parameter may be any
- 5 valid 1394B operating speed (see [tbd]).
- 6 c) Port status. This parameter shall contain the status of the port. The following values are defined for this
- 7 parameter:
- 8 1) DISCONNECTED. The port is currently disconnected.
- 9 2) CONNECTED. The port is connected.
- 10 3) [FAULT. Something has gone wrong with this port.]
- 11 4)

11.3.1.2 CM action request (CM_ACTION.request)

19 The repeater function uses this service to control the connection management function of the port. The CM function shall
 20 service the request immediately upon receipt. The following parameters are communicated via this service:

- 22 a) Action. This parameter shall indicate the action to be taken by the port connection management function. The
 23 following values are defined for this parameter:
- 24 1) SUSPEND.
- 25 2) RESUME.
- 26 3) DISABLE.
- 27 4) ENABLE
- 28 5) NULL (no action)
- 29 b) Port mode. This parameter shall indicate the mode in which the port is permitted to operate. The following values
 30 are defined for this parameter:
- 31 1) DS. Forces the port to attempt connection only in DS mode.
- 32 2) BETA. Forces the port to attempt connection only in Beta mode.
- 33 3) DEFAULT. The port may attempt to connect in any mode.
- 34 4) NULL (no change to port mode)
- 35 c) [others?]

11.3.2 Connection management definition

45 The Connection management is specified as a continuously running C code machine.

47 The functions, variables and constants used in this C code have the following descriptions

53 **Table 11-1—Functions and variables used in Connectivity management**

54 Function, variable or constant	Description
55 Beta_mode	56 Set if the port has determined that it is operating in Beta mode, unset otherwise (i.e. when oper- 57 ating in DS-Mode, or when disconnected)
58 receive_ok	59 In DS_mode indicates the reception of a debounced TpBias signal 60 In Beta_mode, indicates the reception of a continuous electrically valid signal. Note, is set to 61 false during the time that only connection tones are detected in Beta mode
62 local_plug_present	63 Indicates that an external implementation dependent mechanism has determined that there is at 64 least a physical connection from the local node (maybe not connected at the “far end”). Used to 65 avoid performing connection toning if there is definitely no connection. If there is no such mech- 66 anism, then this is set permanently to TRUE

Table 11-1—Functions and variables used in Connectivity management

Function, variable or constant	Description
signal_detect()	Indicates that an electrically valid signal has been detected on TPA since the last call of this function. Does NOT imply reception of valid 8B10B codes or scrambler synchronization.
signal_detect_ok	Used to latch signal_detect() between signal sampling times
TONE_DURATION	TBD (500 microseconds or thereabouts???)
ACTIVE_SAMPLING_INTERVAL	TBD (1 microsecond or thereabouts??)
INACTIVE_SAMPLING_INTERVAL	TBD (30 milliseconds or thereabouts??)

11.4 Node state machine

This is defined as continuously running C code, and maintains a record of whether the nodes is an isolated or boundary node. It shares variables with the port state machine. Note that in this C code, the variables are subscripted, whereas in the port state machine and C code, the same variables are unsubscripted.

Table 11-2—Node state machine

```

void node_status() { // Continuously monitor node status in all states
    int active_ports = 0, i, suspended_ports = 0;

    isolated_node = TRUE; // Remains TRUE if no active port(s) found
    for (i = 0; i < NPORT; i++) {
        if (active[i]) {
            active_ports++; // Necessary to deduce boundary node status
            isolated_node = FALSE; // ALL ports must be inactive at an isolated node
        } else if (connected[i] && !disabled[i])
            suspended_ports++; // Other part of boundary node definition
        boundary_node = (active_ports > 0 && suspended_ports > 0);
    }
}
    
```

11.5 Port State machine

The port connection state machines operate independently for each port. While a port is in the active state its arbitration, data transmission, reception and repeat behaviors are specified by the state machines in XXXX. When a PHY port is in any state other than active it is permissible for it to lower its power consumption; the only functional component of a PHY that shall be active in all states is the physical connection detect circuitry.

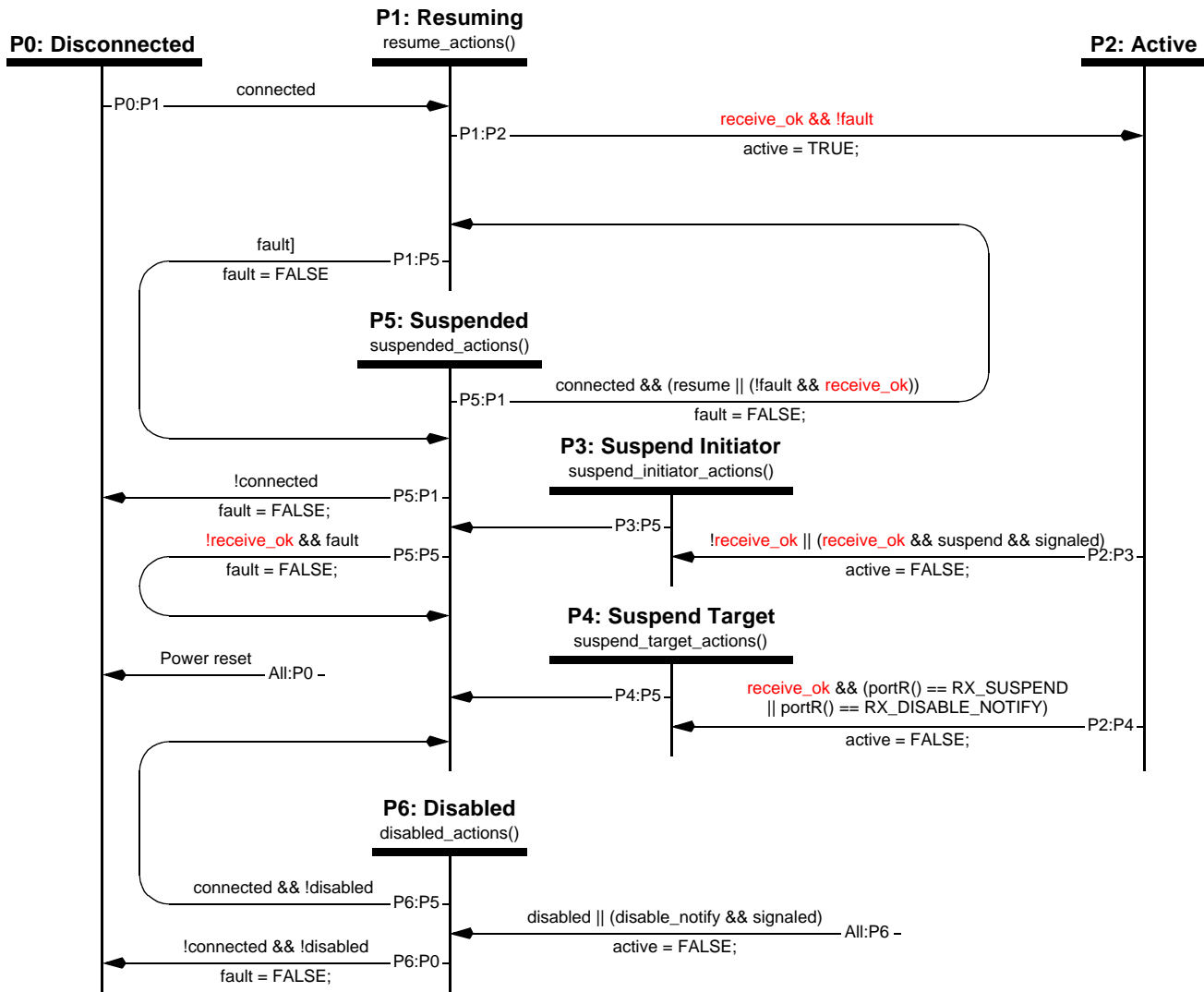


Figure 11-1 — Port connection state machine

11.5.1 Port connection state machine notes

Transition All:P0. A power reset of the PHY initializes each port as disconnected.

Transition All:P6. The local link may immediately disable a port by setting the Disabled bit to one. This transition may also be caused by a remote command packet, in which case the port, if active, shall transmit TX_DISABLE_NOTIFY before the port is disabled.

1 **State P0: Disconnected.** The generation of TpBias is disabled (DS mode) or continuous transmission is stopped (Beta
2 mode) and the outputs are in a high-impedance state. The PHY may place most of the port's circuitry in a low-power con-
3 sumption state. The connection detect mechanism shall be active even if other components of the PHY port are in a low-
4 power state.
5

6
7 **Transition P0:P1.** When a port's connection detect circuitry signals that its peer PHY port is physically connected, the
8 PHY port transitions to the resuming state.
9

10 **State P1: Resuming.** In Beta mode, the PHY commences transmission at the operating speed, and attempts to synchron-
11 ise its receiver. The PHY port tests both the connection status and the presence of `receive_ok` to determine if normal oper-
12 ations may be resumed. If the port is connected, `receive_ok` is set and there are no other active ports, the PHY waits seven
13 RESET_DETECT intervals before any state transitions. Otherwise, in the case of a boundary node with one or more
14 active ports, the PHY waits three RESET_DETECT intervals before any state transitions. A detected bus reset overrides
15 either of these waits, otherwise, when the wait elapses the PHY initiates a bus reset.
16
17

18
19 **Transition P1:P2.** If the PHY port is both connected and `receive_ok` is set, it transitions to the active state.
20

21 **Transition P1:P5.** A resuming PHY port that remains connected to its peer PHY port but faults during the resume hand-
22 shake transitions to the suspended state. The fault condition is cleared so that subsequent detection of `receive_ok` may
23 cause the port to resume.
24

25
26 **State P2: Active.** The PHY port is fully operational, capable of transmitting or receiving and repeating arbitration signals
27 or clocked data. While the port remains active, the behavior of this port and the remainder of the PHY are subject to the
28 cable arbitration states specified in XXXX.
29

30
31 **Transition P2:P3.** Upon the loss of `receive_ok` or the receipt of a PHY remote command packet that sets the `suspend`
32 variable to one, the PHY port leaves the active state to start functioning as a suspend initiator. A loss of `receive_ok` is usu-
33 ally the result of a physical disconnection or the loss of power to the connected peer PHY port. If the transition is the
34 result of a remote command packet, the PHY transmits a remote confirmation packet with the `ok` bit set to one. In the
35 meantime, the suspend initiator has signaled TX_SUSPEND to its connected peer PHY.
36

37 **Transition P2:P4.** If an active port observes an RX_DISABLE_NOTIFY or RX_SUSPEND signal it becomes a suspend
38 target and leaves the active state.
39

40 *Ed:- following needs updating to reflect Beta mode operation*
41

42
43 **State P3: Suspend Initiator.** A suspend initiator waits for `receive_ok` to be zero. If Receive_OK_HANDSHAKE elapses
44 and the connected peer PHY has not driven TpBias low, the suspend operation has faulted and the Fault bit is set to one.
45 In either case the suspend initiator first drives TpBias low for Receive_OK_HANDSHAKE gtime and then places all out-
46 puts in a high-impedance state.
47
48

49
50 **Transition P3:P5.** Upon completion of the actions associated with this state, the PHY port unconditionally transitions to
51 the suspended state.
52

53 *Ed:- following needs updating to reflect Beta mode operation*
54

55
56 **State P4: Suspend Target.** A suspend target sets the `suspend` variable for all the other active ports, which in turn causes
57 them to propagate the RX_SUSPEND signal as TX_SUSPEND. In the meantime the suspend target drives its TpBias out-
58 puts below 0.1 V in order to signal the suspend initiator that RX_SUSPEND was detected. When either the connected
59 peer PHY drives TpBias low or a Receive_OK_HANDSHAKE time-out expires (whichever occurs first), the suspend
60 target disables the generation of TpBias and places the outputs in a high-impedance state.
61
62

63 **Transition P4:P5.** Upon completion of the actions associated with this state, the PHY port unconditionally transitions to
64 the suspended state.
65
66

1 **State P5: Suspended.** The PHY may place most of the port's circuitry in a low-power consumption state. The connection
2 detect circuit. shall be active even if other components of the PHY port are in a low-power state.
3

4 **Transition P5:P0.** A suspended PHY port that loses its physical connection to its peer PHY port transitions to the discon-
5 nected state.
6

7
8 **Transition P5:P1.** Either of two conditions cause a suspended PHY port to transition to the resuming state: a) a nonzero
9 value for the port's *resume* variable or b) the detection of *receive_ok* if the port's Fault bit is zero. A port's *resume* vari-
10 able may be set indirectly as the result of the resumption of other PHY ports.
11

12
13 **Transition P5:P5.** If the port entered the suspended state in a faulted condition (*i.e.*, *Receive_ok* was still present), the
14 fault is cleared if and when *Receive_ok* is removed by the peer PHY.
15

16 **State P6: Disable.** While disabled, the PHY may place most of the port's circuitry in a low-power consumption state. The
17 connection detect circuit. shall be active even if other components of the PHY port are in a low-power state.
18

19
20 **Transition P6:P0.** If the Disabled bit is zero and the PHY port is not physically connected to its peer PHY port, it tran-
21 sitions to the disconnected state.
22

23
24 **Transition P6:P5.** Otherwise, if the Disabled bit is zero and the PHY port is connected it transitions to the suspended
25 state.
26

27 11.5.2 Port connection actions and conditions

28
29 **Table 11-3 — Port connection actions and conditions (Sheet ? of ?)**

```
30
31 int connect_timer, bias_timer;           // global timers
31 int isolated_node, boundary_node;       // global node status variables
32 int connect_detect[NPORT];              // shared with PMD
33 int bias[NPORT];                         // shared with PMD
34 int connect_detect_valid[NPORT];        // flag to track whether connect_detect can be used at a given time
35
36 int active[NPORT]; connected[NPORT]; disabled[NPORT];
37
38 void bias_status() {                    // Continuously running bias update code
39     timer bias_timer;                   // Timer for bias filter
40     boolean filter_bias[NPORT];        // TRUE when applying hysteresis to bias_detect circuit
41     for (i = 0; i < NPORT; i++) {
42         if (bias_filter[i]) {
43             if (bias_detect[i] == bias[i]) // Has bias detect changed since timer started?
44                 bias_timer = 0;           // If so, restart the filtering timer
45             else if (bias_timer >= BIAS_FILTER_TIME)
46                 bias_filter[i] = FALSE;   // Done filtering
47             bias[i] = bias_detect[i];     // Confirm new value in PHY register bit
48         }
49     } else if (bias_detect[i] != bias[i]) { // Detected bias differs from reported bias?
50         bias_filter[i] = TRUE;           // Yes, start a filtering period
51         bias_timer = 0;
52     }
53 }
54 }
55 }
56
57 boolean suspend_in_progress() {         // TRUE if any port suspending
58     int i;
59     for (i = 0; i < NPORT; i++)
60         if (suspend[i])
61             return(TRUE);
62     return(FALSE);
63 }
64
65 boolean resume_in_progress() {         // TRUE if any port resuming
66     int i;
```

```

1     for (i = 0; i < NPORT; i++;)
2         if (resume[i])
3             return(TRUE);
4     return(FALSE);
5 }
6
7 void resume_all_ports() {
8     for (j = 0; j++; j < NPORT)
9         if (!active[j] && !disabled[j] && connected[j])
10            resume[j] = TRUE;        // Resume all other suspended ports
11 }
12
13 // Per port code
14 // The code below runs per port
15 // unsubscripted references to connect_detect, etc are to be interpreted
16 // as references to connect_detect[i], where i is the number of the particular port
17
18 void activate_connect_detect(int delay) {
19     if (Beta_mode) {
20         bport_active = FALSE;
21         ?? any need to wait for anything to happen ???
22
23     } else {
24         tpBias(0);                // Drive TpBias low
25         if (delay != 0) {
26             connect_timer = 0;
27             while (connect_timer < delay) // Enforce minimum hold time for TpBias low
28                 ;
29         }
30         while (!connect_detect)      // Wait for connect_detect circuit to stabilize
31             ;
32         tpBias(Z);                // Release TpBias
33     }
34     connect_detect_valid = TRUE; // Signal OK to connection_status()
35 }
36
37 void send_tone() {
38     pmd_tone_on = TRUE;
39     wait (TONE_DURATION);
40     pmd_tone_on = FALSE;
41 }
42
43 void signal_detect_monitor() { // continuously running - latches signal_detect
44     int sd_detected;          // initialized to FALSE
45     if (~sd_detected) sd_detected = signal_detect;
46 }
47
48 boolean signal_detect_OK() { // true if signal_detect set since we last looked
49     int x;
50     x = sd_detected;
51     sd_detected = FALSE;
52     return(x);
53 }
54
55 void set_beta() { // set beta mode and exchange speed signals to establish operating speed
56     disconnected = FALSE;
57     Beta_mode = TRUE;
58
59     // code to exchange speed signals and set the vairable operating_speed goes in here.
60
61     connected = TRUE;
62     receive_OK = TRUE;
63 }
64
65 void set_DS() { // set DS mode and start up Tp Bias

```

```

1   disconnected = FALSE;
2   connect_detect_valid = FALSE;
3   TpBias(1);
4   connected = TRUE;
5   receive_OK = TRUE;
6   }
7
8   void connection_status() { // continuously running code for each port
9       if (~local_plug_present) { // give up
10          receive_ok = FALSE;
11          Beta_mode = FALSE;
12          connected = FALSE;
13          return; // i.e. to start the routine again;
14      }
15      if (disabled) { // give up
16          if (~Beta_mode) { // look at connection circuit in DS mode
17
18
19          }
20          return;
21      }
22      if (disconnected && local_plug_present) { // look for new connection and determine operating mode
23          int new_connection_detected;
24          new_connection_detected = FALSE;
25          for (i = 0; i < NO_OF_TRIES; i++) {
26              if (bias) break; // don't try to send tones if detect TpBias
27              if (connect_detect && ~new_connection_detected) { // try toning for at least
28                  new_connection_detected = TRUE; // two more times on a new connection
29                  i = NO_OF_TRIES - 2;
30              }
31              if signal_detect_OK {
32                  set_beta();
33                  return; // to next time round in this routine
34              }
35              send_tone();
36              wait (DISCONNECTED_TONE_INTERVAL);
37          }
38          // here if (1) tried toning 10 times without detecting a tone;
39          // (2) detected a connection and tried toning twice without detecting a tone;
40          // (3) detected incoming TpBias
41          if (~connect_detect) return; // to next time round in this routine
42          if (bias) { // still seeing Bias
43              wait(BIAS_HANDSHAKE+10%);
44              if (bias) { // incoming TpBias persists - set DS mode
45                  set_DS();
46                  return;
47              } else { // Bias not present, try toning again
48                  send_tone();
49                  wait(DISCONNECTED_TONE_INTERVAL);
50                  if signal_detect_OK {
51                      set_beta();
52                      return;
53                  } else { // try sending a tone, in case we have powered up whilst the connected
54                      // P1394a PHY was in suspend
55                      connect_detect_valid = FALSE;
56                      TpBias(1);
57                      wait(BIAS_HANDSHAKE);
58                      if (bias) {
59                          set_DS();
60                          return;
61                      } else
62                      activate_connect_detect(0); // includes taking TpBias away
63                  }
64              }
65          } // end of actions whilst disconnected
66

```

```

1 // here if connected
2 if (active) {
3     if (Beta_mode) {
4         wait(ACTIVE_SAMPLE_INTERVAL); // filtering for chatter on SD???
5         receive_OK = signal_detect_OK;
6     } else receive_OK = bias; //DS mode
7     return;
8 }
9 if (resume_in_progress) return;
10
11 // here if suspended - look for disconnect or resume
12 if (Beta_mode) {
13     know_still_connected = FALSE;
14     send_tone();
15     for (i = 0; i < RESUME_CHECKS; i++) {
16         if (signal_detect) {
17             know_still_connected = TRUE;
18             wait(TONE_DURATION);
19             signal_detect; //flush out any residual value from the first detection
20             wait(TONE_DURATION);
21             if (signal_detect) {
22                 receive_OK = TRUE; // to start resuming
23                 return;
24             }
25             wait (RESUME_SAMPLING_INTERVAL);
26         }
27         connected = know_still_connected;
28     } else { // DS mode
29         receive_OK = bias;
30         if (receive_OK) return;
31         connected = connect_detect; // see if still connected
32     } if (~connected) { // disconnection detected
33         Beta_mode = FALSE;
34         if (int_enable && !port_event) {
35             port_event = TRUE;
36             if (link_active && LPS)
37                 PH_EVENT.indication(INTERRUPT);
38             else
39                 PH_EVENT.indication(LINK_ON);
40         }
41     }
42 }
43 void disabled_actions {
44     if (int_enable && !port_event) {
45         port_event = TRUE;
46         if (link_active && LPS)
47             PH_EVENT.indication(INTERRUPT);
48         else
49             PH_EVENT.indication(LINK_ON);
50     }
51     disable_notify = signaled = FALSE;
52     disabled = TRUE;
53     activate_connect_detect(i, 0); // Enable the connect detect circuit
54 }
55
56 void resume_actions() {
57     while (suspend_in_progress()) // Let any other suspensions complete
58         ; // (we may resume those ports later)
59     connect_timer = 0;
60     if ((int_enable || resume_int) && !port_event) {
61         port_event = TRUE;
62         if (link_active && LPS)
63             PH_EVENT.indication(INTERRUPT);
64         else
65             PH_EVENT.indication(LINK_ON);
66     }

```

```

1     }
2     connect_detect_valid = FALSE; // Bias renders connect detect circuit useless, stop toning
3     if (Beta_mode) { // start up and train the port
4         bport_active = TRUE;
5         wait_event(bport_sync_ok);
6         port_synchronized = TRUE;
7     } else {
8         tpBias(1); // Generate TpBias
9     }
10    if (resume == 0 && !boundary_node) resume_all_ports()
11    else
12        resume = TRUE; // Guarantee resume_in_progress() returns TRUE
13    while (((connect_timer < Receive_OK_HANDSHAKE) && !receive_ok) || bus_initialize_active)
14        ; // Wait for peer PHY to generate TpReceive_ok
15    if (receive_ok) { // Connection restored to active state?
16        while ((connect_timer < 3 * RESET_DETECT) && !bus_initialize_active)
17            ;
18        if (!bus_initialize_active) { // No other node initiated reset?
19            if (boundary_node) // Can we arbitrate?
20                isbr = TRUE; // Yes, don't wait any longer
21            else {
22                while ((connect_timer < 7 * RESET_DETECT) && !bus_initialize_active)
23                    ; // Let's wait a little longer...
24                if (!bus_initialize_active)
25                    ibr = TRUE; // Sigh! We'll have to use long reset
26            }
27        }
28    }
29    fault = ~receive_ok; // Resume attempt failed if TpReceive_ok is absent
30    if (fault) // If so, restore usefulness of connect detect circuit
31        activate_connect_detect(i, 0);
31    resume = FALSE; // Resume attempt complete
32 }
33
34 void suspend_initiator_actions() {
35     connect_timer = 0; // Used to debounce receive_ok or for receive_ok handshake
36     if (!suspend) { // Unexpected loss of receive_ok?
37         suspend = TRUE; // Insure suspend_in_progress() returns TRUE
38         if (child) // Yes, parent still connected?
39             isbr = TRUE; // Arbitrate for short reset
40     } else
41         ibr = TRUE; // Transition to R0 for reset
42     while (connect_timer < CONNECT_TIMEOUT)
43         ; // Time for receive_ok to stabilize
44 }
45 while ((connect_timer < Receive_OK_HANDSHAKE) && receive_ok)
46     ; // Wait for suspend target to deassert receive_ok
47 fault = receive_ok; // Suspend handshake refused by target?
48 activate_connect_detect(i, Receive_OK_HANDSHAKE); // Also guarantees handshake timing
49 }
50
51 void suspend_target_actions() {
52     int j;
53     if (resume_in_progress()) // Other ports resuming?
54         resume = TRUE; // OK, do suspend handshake but resume afterwards
55     suspend = TRUE; // Insure suspend_in_progress() returns TRUE
56     if (portR() == RX_DISABLE_NOTIFY) { // Is our peer PHY going away?
57         breq = IMMEDIATE_REQ; // Topology change! Reset on other (active) ports
58         isbr = TRUE;
59     } else if (portR() == RX_SUSPEND && !resume) { // Don't propagate if resume in progress
60         for (j = 0; j < NPORT; j++)
61             if (active[j]) // Otherwise all active ports become suspend initiators
62                 suspend[j] = TRUE;
63         breq = IMMEDIATE_REQ; // Invoke transmitter to propagate TX_SUSPEND
64         isbr = TRUE; // Alert link that we're now isolated
65     }
66 }

```

```

1   while (portR() == RX_DISABLE_NOTIFY || portR() == RX_SUSPEND)
2       ; // Let signals complete before receive_ok handshake
3   activate_connect_detect(i, Receive_OK_HANDSHAKE);
4   }
5
6   void suspended_actions() {
7       signaled = suspend = FALSE;
8       if (int_enable && !port_event) {
9           port_event = TRUE;
10          if (link_active && LPS)
11              PH_EVENT.indication(INTERRUPT);
12          else
13              PH_EVENT.indication(LINK_ON);
14      }
15  }

```

11.6 Rationale (informative)

Connection detection

When the port is disconnected, the procedure described in this section aims to detect a reconnection, then to determine the appropriate mode of operation (DS mode or Beta mode) and, if Beta mode, the appropriate operating speed. The same underlying mechanisms are used during suspend to determine that a port has become disconnected. In both cases, the emphasis has been to provide a very low power mechanism which meets all the appropriate constraints. A simplified algorithm applies if the port is Beta only capable.

P1394b uses connect_detect_valid to drive a toning scheme. A tone is transmitted on TPB. A signal_detect circuit listens on TPA. (The output on TPA is set to high impedance). The signal_detect function returns TRUE if an electrically valid signal has been detected on TPA since the last call of the function. The signal_detect function does not provide any guarantee of quality of signal, and does not imply scrambler synchronization or the reception of valid 8B10B codes.

The design of the algorithm which is used during disconnection and during suspend in order to detect a change in port status is based on the following fundamental points, given with the reasoning on each.

#1. Can't tone and drive TpBias simultaneously. Proof: P1394a and earlier PHY's would attempt to interpret the tone if TpBias were present. Only safe way to tone into a p1394a mode is when TpBias is deasserted.

#2. Can't wait for remote port on peer PHY to initiate TpBias. Proof: A P1394a node w/ a suspended port won't initiate TpBias as long as the DC connection status is continually active. Consider a P1394a PHY and a P1394b PHY which are connected via a suspended link. Subsequently, if the P1394b experiences a power-on reset (device powered off for a while and then turned back on), it needs to re-establish connectivity with the P1394a node. If the eager beta algorithm waited for the P1394a device to initiate TpBias ... it would never come. Consequently, at least one step in eager Beta must lead to the initiation of a TpBias handshake.

#3. Can't assume initial receipt of TpBias means a non-P1394b device is attached, and must listen for tone even when driving TpBias. Proof: If a P1394b port (called X) tries toning first and hears nothing, then according to #2 above, it will have to initiate TpBias. If a connected P1394b port (called Y) on a peer PHY then powers up, it will detect TpBias without a tone. Thus, assuming the presence of TpBias always indicates a P1394a port leads to a false detection of the remote peer. To avoid this, port Y should tone first even if it sees TpBias (courtesy of Mr. LaFollette). Port X, which is still driving TpBias, must still listen for a tone. After detecting the tone from Port Y, Port X will revert to toning and the beta mode handshake can commence.

#4. During upstarting, must initiate toning with some frequency. Proof: Assume an AC path between two P1394b ports and allow that the ac path can have passive connection points (RJ-45 jack on the wall or in the patch panel, a optical wall socket, etc.). If the passive connection is not in place, the two P1394b nodes at power-on reset will attempt to initiate a

1 handshake. Both progress to the TpBias assertion phase mandated in #2 above without ever detecting the remote port. If
2 the eager Beta scheme stopped at this point with the continuous assertion of TpBias, a later connection at the passive
3 point won't be detected by either node. For this reason, the eager Beta procedure must attempt toning with some fre-
4 quency to allow for passive connections. (A plug present feature at the PHY doesn't handle the case of passive connec-
5 tion points in-between the attached ports.) This suggests that the eager Beta sequence would need to alternate between
6 toning and generating TpBias. Alternatively, we might be okay with a single phase of TpBias and then revert to contin-
7 uously toning.
8

9
10 #5. Can't listen for a tone when generating TpBias. Proof: When generating TpBias, the remote node may be a 1394-1995
11 node (or indeed a P1394a node). Such a node will interpret the TpBias signal as a connection, and will start sending arbi-
12 tration signals (such as reset). These will trip the "signal-detect" mechanism, which operates by looking for a differential
13 signal on TPA.
14

15
16 Corollary to Axiom 5: A Beta-PHY doesn't need to send a tone when receiving TpBias because the Beta-PHY on the
17 other end of the cable that's generating TpBias can't listen for the tone anyway.
18

19
20 P1394b uses connect_detect_valid to drive a toning scheme. A tone is transmitted on TPB. A signal_detect circuit listens
21 on TPA. TPA is set to high impedance. The signal_detect() function returns TRUE if an electrically valid signal has been
22 detected on TPA since the last call of the function. The signal_detect function does not provide any guarantee of quality
23 of signal, and does not imply scrambler synchronization or the reception of valid 8B10B codes.
24

25
26 A 1394-1995 node at the far end will assert TpBias on its TPA, but will not see TpBias on its TPB, and so will not think
27 it is connected. A P1394a node on the other end will sense con_status and start its debounce timeouts.
28

29
30 Based on the above, the algorithm used following power on reset or at any time we are trying to determine connection
31 status is an "eager Beta mechanism", and operates as follows:-
32

33 1) Begin toning. If tone is heard, then beta mode is possible. If TpBias is detected, then go straight to 2, otherwise stay
34 in this state until 10 (???) tones have been generated. If a change from disconnected to connected in connect_detect
35 is detected, then tone for two more times, then go to 2.
36

37 2) Check for "connect_detect". If not set, then go back to the start of 1. Otherwise, check for TpBias. If TpBias is
38 detected, wait a period of time (BIAS_HANDSHAKE - or, perhaps, BIAS_HANDSHAKE + "A little More") and then
39 sample Bias once again.
40

41
42 If Bias is still present, assert TpBias and DS mode is established.
43

44
45 If Bias is NOT present, generate the startup tone. If the attached port is an A-PHY or a 1394-1995 PHY, no response will
46 occur, otherwise, an attached B-PHY(1) will respond with the startup tone and Beta-Mode will be established.
47

48
49 If no response is received as a result of generating the startup tone, asserts TpBias for BIAS_HANDSHAKE. If Bias is
50 then detected, DS mode is established, otherwise, when no response, B-PHY(0) resumes alternating between toning and
51 asserting TpBias.
52

53
54 A 1394-1995 port at the far end will assert TpBias on its TPA, but will not see TpBias on its TPB, and so will not think
55 it is connected. A P1394a node on the other end will sense con_status and start its debounce timeouts.
56

57
58 If an P1394a port is the connection at the other end, and, if just by chance, the local P1394b port does not respond with
59 TpBias before the P1394a port "times-out" there really is no issue because the P1394a port will enter into a resume-fault
60 condition, but, because the P1394b port did not receive a tone in reply to its last tone attempt, it will generate TpBias
61 again causing the P1394a port to "wake" and become a resume target.
62

63
64 In the instance that the far port is a 1995 port there will not ever be an issue because it will always be generating TpBias
65 and when the P1394b port finally generates TpBias it (the B-PHY) will eventually generate a bus-reset because of the
66 "resume" process if a bus reset is not detected from somewhere else.

The set of possible port connections that a bilingual port may be connected to are:-

Figure 11-2—Connect Status value in various connection scenarios

	Connect_status	tone exchange	action
No connection	No	fails	
DC connection to P1394a	Yes	fails	set DS
DC connection to bilingual	Yes	succeeds	set Beta
DC connection to Beta only	probably	succeeds	set Beta
AC connection to bilingual	No	succeeds	set Beta
AC connection to Beta only	No	succeeds	set Beta
DC connection to optical transceiver	No - must be biased so as not to trigger the Con detector	fails	
AC connection to optical transceiver	No	fails	

In the case that the answer is yes, the connected flag is set to true.

Note that the tone transmission and detection continues through the debounce period. If a tone is detected during this time, then Beta mode is set.

Once a Beta mode connection has been determined, a speed negotiating procedure is used. A single protocol is used to negotiate the speed anywhere in the range S100 to S3200.

Upon connection, the only property of the connecting medium that can be relied upon is that it can support the exchange of 125MHz tones provided they last at least 500 microseconds. This is true until continuous operation is established, (particularly, for example, because of the start-up latencies of the optical components). The aim is to establish continuous operation at the correct operating speed, and to avoid any need for matching configuration options at "both" ends to determine what a "common denominator" operating speed should be.

These aims are achieved by using the same tone as is used for connection detection, differentiating where necessary between the occasional tone to identify connect/disconnect events, a pattern of tones to negotiate the operating speed, and a continuous tone (in suspend) to indicate that it is time to resume. For this latter, we do indeed know the operating speed, but there seems to be no good reason for making this indication any different from the tone used for connect detect apart from its duration (remember, we only want a simple signal detect circuit to be alive during suspend).